intel

ARTICLE REPRINT

AR-272

April 1983

Bubble-Memory Support Chips Allow Tailored-System Design

Richard Pierce
Applications Engineer
Intel Corporation

Reprinted with permission from EDN, Volume 20, No. 7. Copyright Cahners Publishing Company, 1983.

APRIL 1983
ORDER NUMBER: 230645-001

Bubble-memory support chips allow tailored-system design

Using special support chips gives you flexibility in designing a bubble-memory system. And understanding design tradeoffs helps configure a system that best suits your needs.

Richard Pierce, Intel Corp

Designing a bubble-memory system—with its advantages of small size, high reliability and nonvolatility—is easy when you use system support devices. Such a family of LSI chips allows you to tailor a system to meet requirements on specs such as access time and power consumption and to modularly expand the system for increased storage capacity. How you configure and use a bubble-memory system involves tradeoffs, though, and balancing those tradeoffs requires knowledge of the intended system application. This article discusses bubble-system tradeoffs and other design considerations and presents a specific design using Intel's 1M-bit bubble-memory device and family of support chips.

Interface appears as a peripheral controller

The devices available for building a 1M-bit bubble-memory system are the 7110 magnetic-bubble module, the 7220-1 bubble-memory controller, the 7242 formatter/sense amplifier, the 7230 current-pulse generator and the 7250 coil predriver (see **box,** "Bubble-memory devices"). Communication with the 7220-1 controller **(Fig 1)** is the key function, because this device provides the bubble memory's only interface with the outside world. It allows you to interface with a bubble-memory system through a standard μP bus just as with any peripheral. Software directs the controller to choose one of three memory-access methods: direct memory access (DMA), interrupt or poll.

The 7220-1 bubble-memory controller (BMC) has several functional sections. The system-bus interface provides an asynchronous interface to a host processor, transferring 12.5k bytes/sec with a 4-MHz system clock. The controller also has a 40-byte first-in first-out (FIFO) memory buffer through which data passes on its way to the 7242 formatter/sense amplifier (FSA). This FIFO's primary purpose is to reconcile timing differences between the user and the FSA, because the system bus typically can transfer data much more rapidly than the bubble memory can.

The BMC's DMA and interrupt logic handle data transfers. In addition, an internal register file contains six user-accessible 8-bit registers: a command register, a status register and four registers that store information pertaining to the system's operational mode and configuration. The command register accepts user-issued codes that initiate data transfers, and the status register indicates the BMC's current state.

The remaining functional sections of the BMC

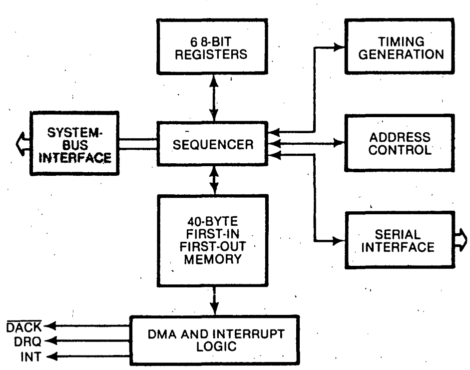


Fig 1—A bubble-memory controller, *which interfaces to a host μP through a system bus, generates all support-circuit timing and control signals, maintains memory-address information and performs data conversions for transfers to a bubble-memory module.*

Communicate with a bubble memory as with a peripheral controller

generate all the support-circuit timing and control signals, maintain memory-address information and perform parallel-to-serial and serial-to-parallel conversions for transfers to the bubble-memory module.

Interface circuitry depends on transfer mode

The type of circuitry you design to interface with the BMC depends on the mode of data transfer chosen. For a polled implementation, the requirements reduce to interfacing to a standard μP bus. One such interface design, for the 8088, appears in **Fig 2;** it consists of address-decode logic, data-bus-decode and buffering logic, a clock circuit and miscellaneous control logic. The clock circuit must provide a 4-MHz (±0.1%) system clock with a 50% (±5%) duty cycle.

Fig 2's system operates from 12 and 5V only, and one important part of this design is its automatic power-fail circuitry **(Fig 3),** which monitors these voltages. An important aspect of bubble-memory devices is that the coil drive current (which moves bubbles around within the device) must always have the proper phase and amplitude, without transients, to ensure data integrity. **Fig 3**'s power-fail circuit prevents transients and—when voltages drop 6% below normal level—stops the coil currents while maintaining the proper phase. You can also expand the power-fail circuit to include recommended features such as ac power-fail and ac or dc overvoltage protection.

Bubble-memory devices

Several different devices **(figure)** make up a bubble-memory system. At the heart of the system is the 7110 magnetic-bubble memory (MBM), with a user data capacity of 1M bits (128k bytes).

This chip embodies a major-track/minor-loop architecture, in which bubbles serially propagate into and out of the memory tracks and get stored in minor (storage) loops (EDN, September 1, 1982, pg 198). This organization permits creation of redundant storage loops, increasing device yield by tolerating defects in as many as 15% of the loops. Testing by the manufacturer detects the unusable loops and writes a map—showing the usable loops—into an additional storage loop called a boot loop. The map also appears on the 7110's label.

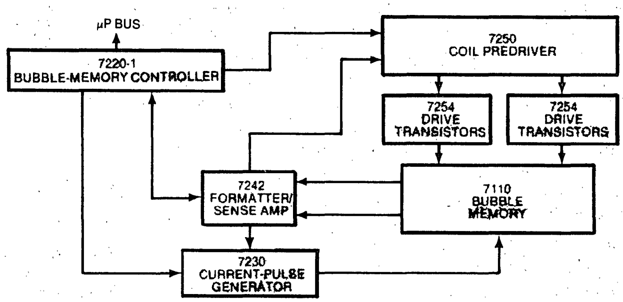
Internally, the 7110 consists of two identical 512k-bit sections. Although these sections are essentially independent, they operate simultaneously to give a factor-of-two data-rate improvement.

User interface to the system occurs through a 7220-1 bubble-memory controller (BMC). This device provides the system-bus interface, performs serial-to-parallel and parallel-to-serial data conversions, generates all timing and control signals necessary for proper operation of support circuitry and interprets and executes user requests for data transfers. The BMC's interface makes the bubble-memory system look like a peripheral to a μP-system bus.

The 7242 formatter/sense amplifier (FSA) interfaces independently to each half of the bubble memory. Its integrated sense amplifier accepts low-level voltage signals from the MBM's bubble detectors during read operations, and the device also performs data-formatting tasks that include the transparent handling of the MBM's redundant loops. In addition, the FSA sends TTL-level control signals to the 7230 current-pulse generator (CPG) during write operations.

The 7230 supplies the current pulses that generate bubbles in the MBM and transfer them into and out of the storage loops. The CPG's integrated power-fail-detection circuitry initiates an orderly shutdown of the current sources when power fails.

The 7520 coil predriver (CPD) interfaces the 7220-1 BMC to the two 7254 drive transistors, which supply the relatively high peak currents required by the 7110 MBM's X and Y coils. These currents induce the in-plane rotating magnetic field that moves the magnetic bubbles.



Support chips *allow the design of a complete bubble-memory system around Intel's 7110 magnetic-bubble memory device.*

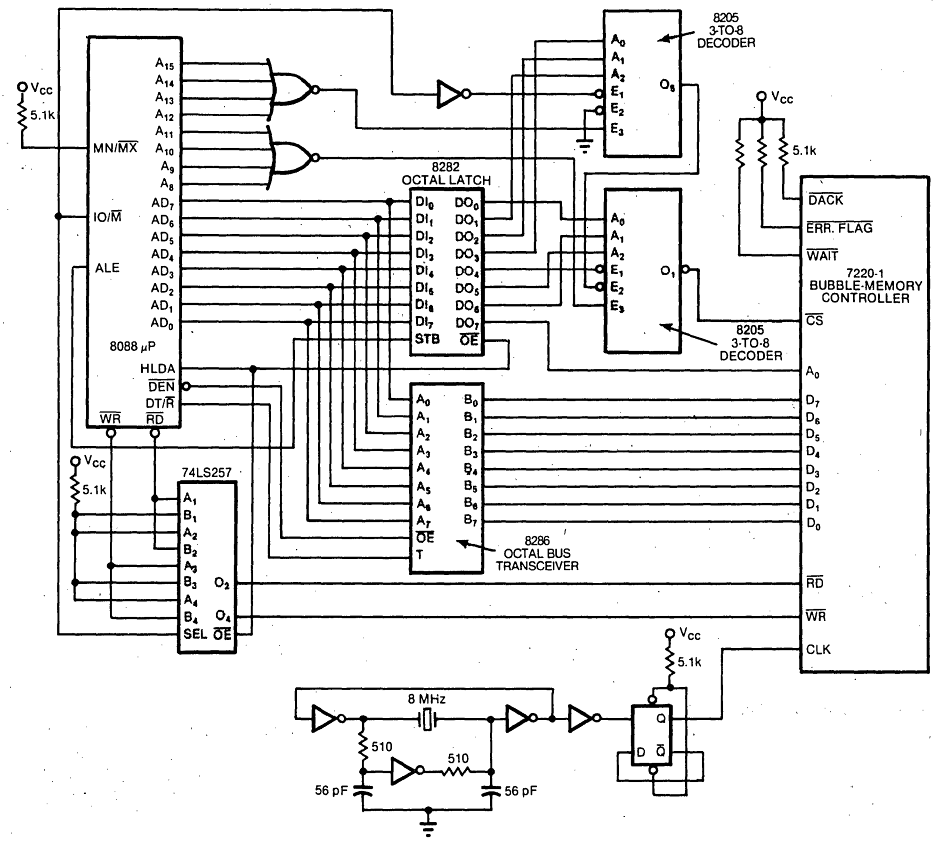


Fig 2—**Interfacing a bubble memory** *can be almost as simple as interfacing a standard μP bus. This interface connects a 7220-1 bubble-memory controller with an 8088 μP.*

Guaranteeing power-supply requirements is an important part of the design process. The supply voltages must be within 5% of their specified values, and the power-off/power-fail decay rates (**Table 1**) must allow 150 μsec max for an orderly shutdown and reset of all support circuits. No restrictions apply to voltage rise times or sequencing.

A simple calculation determines your system's required storage capacitance to achieve **Table 1**'s voltage decay rates. The worst-case power-supply capacitance requirement is

$$\frac{Q_{MAX}}{V_{MIN}} = \frac{\Delta I_{MAX} \Delta T_{MAX}}{\Delta V_{MIN}}$$

Typical capacitance values for a system with one bubble-memory module, excluding any additional current drain from unrelated circuitry, are 805 and 350 μF for the 5 and 12V supplies, respectively.

Another important design factor in custom bubble-memory boards is careful circuit layout to minimize interference from the 7110's large drive signals with

Choose a memory-access method: direct, interrupt or poll

nearby small sense signals. The pin assignments of the 7110 and its support devices optimize board layout and maximize circuit density, and the package layout used in Intel's Bubble Prototype Kit (BPK-72) helps ensure error-free operation. As shown in **Fig 4,** this layout places all support circuits near the 7110; note particularly that the 7110 and the 7242 FSA must not be physically separated.

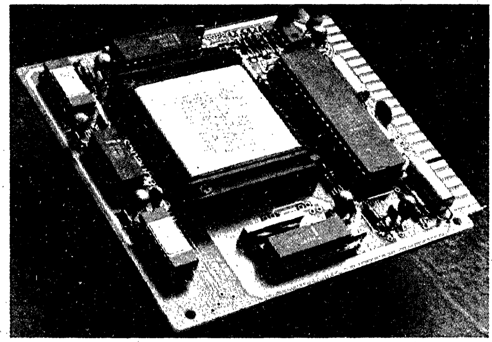


Fig 4—Proper component layout *helps ensure error-free bubble-memory operation. In this Intel Bubble Prototype Kit (BPK-72), all support circuits are close to the 7110 memory module.*

Software controls the system

Software is a vital part of bubble-memory design, too; it's the major contributor to a system's efficient and reliable operation. Software interface modules (often called bubble drivers) accept processor-issued commands and control and monitor command execution; they also return status information to the processor.

Software drivers depend on the mode of data transfer: interrupt-driven I/O, DMA or polled. In a DMA implementation, the software need only set up the DMA controller's memory-address and transfer counts and initiate the data transfer; Intel's 8257 DMA-controller hardware automatically handshakes with the μP and BMC to perform each transfer.

TABLE 1 — BUBBLE-MEMORY POWER-SUPPLY REQUIREMENTS

VOLTAGE	MARGIN	POWER-OFF/POWER-FAIL DECAY RATE
12V	±5%	< 1.10 V/mSEC
5V	±5%	< 0.45 V/mSEC

In the interrupt mode, on the other hand, the software is responsible for performing memory-read and -write operations to transfer 22-byte data blocks to and from the BMC's FIFO on receipt of an interrupt. A BMC output signal—typically wired as a second-level interrupt—indicates when the BMC's FIFO becomes half full (during read operations) or half empty (during write operations).

The third I/O method—polling—is similar to the interrupt-driven mode except that in it, data moves one byte at a time. The software determines when to transfer data by continually polling a bit in the BMC's status register. This status bit indicates the presence or

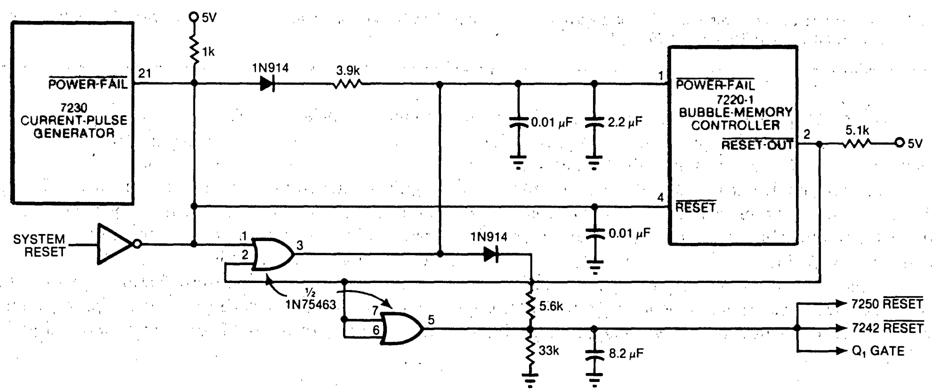


Fig 3—Power-fail circuitry *monitors the 12 and 5V supplies for* **Fig 2**'*s circuit and maintains data integrity in case of power failure by stopping coil currents in the proper phase.*

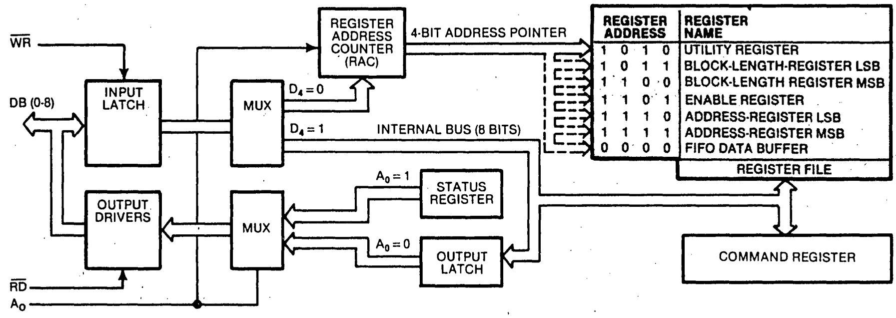


Fig 5—Software accesses the bubble-memory controller *through a system-bus interface consisting of two I/O ports. A ONE on address line A_0 selects the command/status port, and a ZERO selects the bidirectional data port.*

absence of data—to be read or written by the host processor—in the BMC's FIFO.

Although the polling mode is simple to implement, its software requirements are the most demanding. Because data transfers one byte at a time, the software must continually monitor the status register to ensure that the FIFO doesn't underflow or overflow with data.

Each of these data-transfer modes has unique advantages that must be weighed with each particular application. The DMA mode, for example, permits the processor to continue executing instructions while a transfer is in progress. The interrupt and polled modes, on the other hand, offer lower cost but are often too slow in systems incorporating multiple bubble devices connected in parallel. Eight 7110 bubble modules in parallel can transfer data at rates as high as 100k bytes/sec, and these high-performance systems normally use the DMA mode.

Understand protocols and definitions

Developing the software drivers for any bubble-memory system requires a clear understanding of command protocols and register definitions. The software communicates with the bubble-memory controller through the system-bus interface, consisting of two I/O ports selected by the state of the least significant address line **(Fig 5)**. When A_0 is ONE (active), the command/status port gets selected; when it's ZERO (inactive), the bidirectional data port gets selected.

The command/status port serves a dual function: Reading the port accesses the status register, and writing to the port accesses a register determined in part by data bit D_4. This register is the command register if D_4 is ONE; it's one of the six parametric registers or the BMC's FIFO if D_4 is ZERO. In the latter case, the contents of a register-address counter (RAC) specify one of the seven possibilities.

The command register accepts one of 16 interface commands. The initiation of nondata transfers occurs merely by writing the proper command code to the command register, while a data transfer (initialize or read/write) requires prior loading of the parametric registers. Those registers specify the operating mode and system configuration, plus the data transfer's starting address and length (in 64-byte pages). After loading of the command register, the BMC automatically executes the command. An indication of success or failure returns in the status register.

An autoincrementing feature in the RAC facilitates loading the parametric registers before data-transfer commands. After the processor loads a value—specifying a particular parametric register—into the RAC via the command port, the next write operation to the data port accesses that register. Each write operation also increments the RAC, so each subsequent operation accesses the next register in sequence. After incrementing to address 0 (the BMC's FIFO), however, the RAC stops incrementing and continues to point to the FIFO until modified by software.

Software drivers perform data transfers

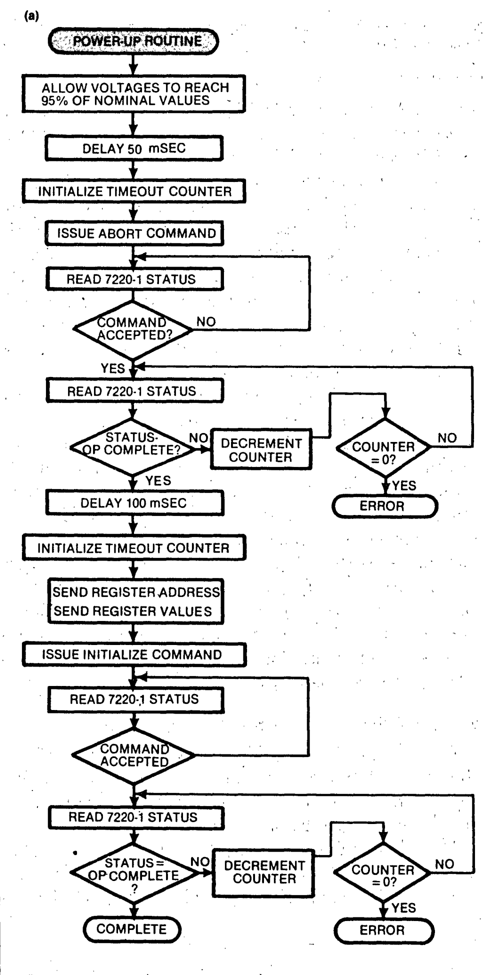
An example set of BMC software drivers **(Fig 6)** shows how data-transfer operations occur in the polled mode. The three drivers perform a power-up procedure, write registers, and read and write data.

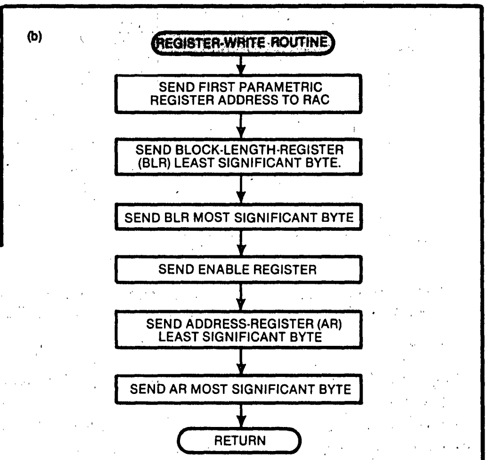
The power-up procedure **(Fig 6a)** enables bubble-memory support devices in an orderly fashion and permits subsequent reading and writing of memory

Meet power-supply requirements to ensure reliable operation

pages. The first command it issues is Abort; the host then loads the parametric registers with appropriate values for a 1M-bit system configuration and issues an Initialize command. During command execution, the host processor constantly polls the BMC's status register to determine when the command finishes. An additional status-register check determines whether the command and/or data transfer is successful.

Successful completion of the Initialize command





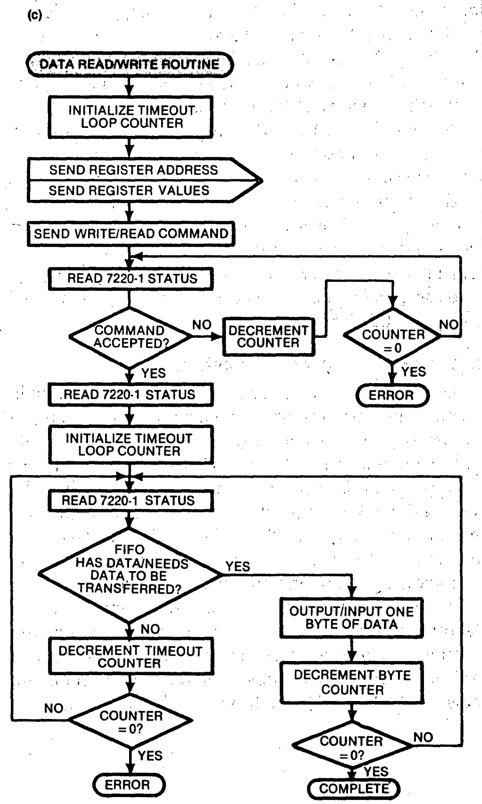


Fig 6—Bubble-memory software drivers *perform power-up* ***(a)****, write registers* ***(b)*** *and read and write data* ***(c)****.*

Proper physical layout helps reduce errors

indicates that the system is ready to transfer data. To initiate a data transfer, the host loads the parametric registers (**Fig 6b**) with the memory-page address and number of pages to transfer and then issues a read or write command (**Fig 6c**). This command transfers data between the system bus and the BMC's FIFO at a rate of 80 μsec/byte, excluding access time. The software polls the status register to determine when to transfer each byte and also maintains a count of the bytes transferred. In systems operating in the DMA or interrupt mode, however, maintaining this count is unnecessary.

Of course, all software driver routines should contain error-handling capabilities, but the examples shown here ignore these capabilities for the sake of simplicity. Numerous error-correction options exist in a bubble-memory system, though, and all are selectable under software control. As an example, the 7242 formatter/sense amplifier uses a 14-bit error-correction code with each 256-bit block of data. This code can correct all single-error bursts of five bits or less, improving the data error rate by several orders of magnitude while remaining user transparent.

Expand memory in modules

Having considered a bubble-memory-system design, turn to memory-expansion considerations. One BMC can control as many as eight bubble-memory modules, and with multiple BMCs, you can configure even larger systems (**Fig 7**). A memory module—providing expansion in increments of 128k bytes—consists of a 7110 bubble device, one 7230 current-pulse generator, one 7250 coil predriver, a 7242 formatter/sense amplifier and two 7254 drive-transistor packages.

Expansion can occur in three ways. The first approach uses the BMC's built-in ability to handle multiple bubble devices. This scheme relies on the BMC to time-slice the serial bus between the BMC and the appropriate number of 7242 FSAs in the system and to output appropriate control signals to the 7242, the 7230 and the 7250. Data flow in this expanded system is similar to that in a single-module system.

A second expansion approach takes advantage of provisions in the BMC for paralleling controllers. This approach provides a greater word width at the system bus and still allows each BMC to accommodate as many as eight bubble devices.

A third option switches banks of bubble devices into or out of a circuit under external control. (Switching is possible because each support device has a chip-select pin.) The maximum number of devices in each bank remains eight, but the number of banks is unlimited. You can even multiplex entire subsystems of this type, because the BMC chip itself has a chip-select input.

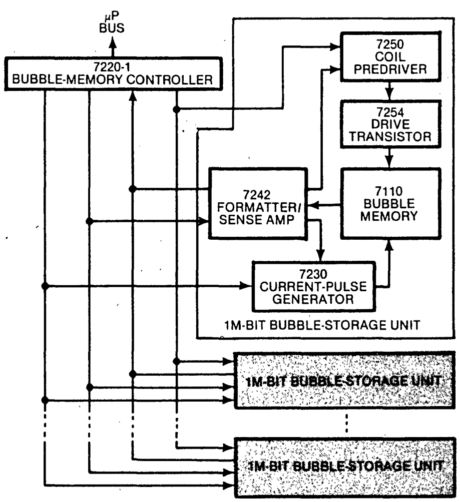


Fig 7—Bubble-memory expansion *occurs in increments. Each 7220-1 controller can handle as many as eight 128k-byte modules.*

Nonvolatility permits power savings

Your design of a bubble-memory system can also take into account the system's nonvolatility in order to minimize power consumption. Consider, for example, a bubble system for a portable terminal that stores inventory and sales figures and periodically transmits them to a central computer. The bubble memory in such a terminal requires power only during memory access, so you can remove power from the bubble system during much of the time that the terminal is in use.

In some applications—for example, systems that only occasionally transmit a small amount of data—you can also reduce power consumption by using a faster initialization scheme. The usual initialization procedure, which reads boot-loop information from a bubble device to identify redundant storage loops, requires as much as 160 msec per device—a significant percentage of power-on time for such systems. If you store the boot-loop information in EPROM, though, you can download it from there much more rapidly and thus with less power.

Design tradeoffs occur at system level

Still other design considerations exist at the system level; design tradeoffs concern such factors as memory capacity, access time, data rate and power consump-

Software interface modules contribute to system efficiency

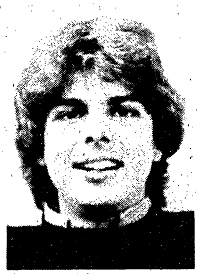
TABLE 2 — BUBBLE-MEMORY PERFORMANCE PARAMETERS

	ONE MBM	FOUR MBMs	EIGHT MBMs OPERATED IN PARALLEL	EIGHT MBMs MULTIPLEXED ONE AT A TIME
CAPACITY	128k BYTES	512k BYTES	1M BYTES	1M BYTES
NOMINAL DATA RATE	68 kHz	272 kHz	544 kHz	68 kHz
AVERAGE ACCESS TIME	48 mSEC	48 mSEC	48 mSEC	48 mSEC
POWER DISSIPATION (100% DUTY FACTOR)	6W	20W	40W	11W
STANDBY POWER	1.55W	3.7W	7.0W	7.0W
BOARD AREA	16 IN.²	45 IN.²	90 IN.²	90 IN.²

tion. As **Table 2** shows, the 7110 bubble module's access time averages 48 msec (7.4 msec best case, 80 msec worst case), independent of the number of modules in a system. You can increase the data rate, however, by operating the devices in parallel. The approach requires more power, but using eight 7110s instead of one increases the nominal bit rate from 68 to 544 kHz.

Finally, you can improve the overall data rate by reducing the average time required to access a memory page. The access time for any single page is random, but no access delay occurs for succeeding pages. Thus, in time-critical applications where successive page accesses aren't random, least recently used (LRU) and lookahead algorithms can help reduce page-access time. **EDN**

Author's biography



Richard Pierce works in Intel's Nonvolatile Memory Div (Santa Clara, CA), supplying customer support and developing new applications. He holds a BSEE degree from Purdue University and previously worked for Cincinnati Electronics. Dick is a member of the IEEE, and his hobbies include biking, photography and skiing.